Lista de Exercícios 02

Marcos Daniel Souza Netto

November 29, 2023

1 Questão 1

1. Inserção é, em geral, mais eficiente que o Quicksort para arquivos quase ordenados

RESPOSTA: Verdadeiro, normalmente isso ocorre porque o Insertion Sort é adaptável, e nesse caso, ele executa trocas pontuais (na maior parte das vezes temos que o elemento a ser inserido no subvetor ordenado já está na posição correta).

2. O algoritmo de pesquisa em uma árvore AVL é o mesmo de uma árvore binária de pesquisa.

RESPOSTA: Verdadeiro, a diferença entre as duas árvores é que a AVL é balanceada, e a árvore binária de pesquisa não. Então a pesquisa numa BST pode ter, no pior caso, complexidade $O(n)$, enquanto que na AVL, a complexidade é $O(log(n))$.

3. A complexidade de inserção em uma árvore AVL tem o pior caso igual a O(n).

RESPOSTA: Falso, a complexidade de inserção em uma árvore AVL tem o pior caso igual a $O(log(n))$.

4. O Bucket Sort tem complexidade $O(n^2)$ quando o número de buckets tende a n.

RESPOSTA: Falso, o Bucket Sort tem complexidade $O(n)$ quando o número de buckets tende a n. Lembre do caso do TP2, quando o número de buckets era igual ao número de elementos do vetor, a complexidade era $O(n)$. Mas de forma geral temos que a complexidade de tempo do Bucket Sort é $O(n + k)$, onde n é o número de elementos do vetor e k é o número de buckets.

5. O Counting Sort requer uma memória auxiliar de tamanho n.

RESPOSTA: Falso, o Counting Sort requer uma memória auxiliar de tamanho k, onde k é o maior elemento do vetor. Complexidade de tempo: $O(n+k)$ (Caso tenhamos n maior que k ou k maior que n). Complexidade de espaço: $O(k)$.

7. O uso de um sentinela na pesquisa sequencial reduz a sua ordem de complexidade

RESPOSTA: Falso, lembrando que a sentinela utilizada na pesquisa sequencial é um elemento que não pertence ao vetor, e é utilizado para que não seja necessário verificar se o vetor chegou ao fim. A complexidade continua sendo

$O(n)$, estamos diminuindo somente a constante. O certo seria a diminuição do custo do algoritmo, e não da complexidade.

8. O melhor caso de uma pesquisa em tabela hash será $O(1)$, independente do tratamento de colisão adotado.

RESPOSTA: Verdadeiro, o melhor caso de uma pesquisa em tabela hash ocorre quando não temos problema de colisão, e a primeira posição que calculamos é a posição do elemento que estamos procurando.

2 Questão 6

Para fazer.

3 Questão 7

a. Qualquer algoritmo de ordenação realiza pelo menos $\Omega(n \cdot \log n)$operações

RESPOSTA: Falso, o algoritmo de ordenação Counting Sort realiza $O(n+k)$ operações, onde k é o maior elemento do vetor. Ou seja, ordem linear de tempo. Ou até mesmo o Insertion Sort, que realiza $O(n)$ operações no melhor caso.

b. Não conhecemos nenhum algoritmo de ordenação cujo pior e o melhor caso sejam idênticos em termos de complexidade assintótica

RESPOSTA: Falso, podemos citar o Heap Sort que em qualquer caso tem complexidade $O(n \cdot \log n)$.

c. Insertionsort e Selectionsort são exemplos de algoritmos de ordenação estáveis

RESPOSTA: Verdadeiro, ambas as ordenações conservam a ordem relativa dos elementos iguais.

d. QuickSort com escolha do pivô realizada através de sorteio, com todas as posições do vetor de entrada podendo ser escolhidas com mesma probabilidade, é um algoritmo ótimo de ordenação.

RESPOSTA: Falso, o QuickSort se torna ótimo quando o pivô é escolhido como o elemento do meio do vetor/subvetor.

e. Um algoritmo de ordenação que realiza comparações e tem complexidade de $O(n \cdot \log n)$ é um algoritmo ótimo.

RESPOSTA: Em geral os algoritmos rápidos de ordenação que realizam comparações tem complexidade $O(n \cdot \log n)$, mas não há prova de que sejam ótimos.

4 Questão 9

Vamos supor que temos um arranjo de n registros de dados para ordenar e que a chave de cada registro tem valor 0 ou 1. Um algoritmo para ordenar tal conjunto de registros poderia ter algum subconjunto das três critérios desejáveis a seguir:

1. O algoritmo é executado com complexidade de $O(n)$.

2. O algoritmo é estável.

3. O algoritmo ordena localmente, sem utilizar mais que uma quantidade constante de espaço de armazenamento além do arranjo original.

a. Indique um algoritmo que satisfaça os critérios 1 e 2.
b. Indique um algoritmo que satisfaça os critérios 1 e 3. c. Indique um algoritmo que satisfaça os critérios 2 e 3. d. Algum dos seus algoritmos de ordenação dos itens (a)-(c) pode ser usado para ordenar n registros com chaves de b bits usando radix sort com complexidade O(bn)? Explique como ou por que não